Combining Checkpointing and Replication for Reliable Execution of Linear Workflows

Anne Benoit, Aurélien Cavelan, Florina Ciorba, Valentin Le Fèvre, Yves Robert

Combining Checkpointing and Replication for Reliable Execution of Linear Workflows

Anne Benoit*‡, Aurelien Cavelan†, Florina M. Ciorba†, Valentin Le Fèvre*, Yves Robert*§
*ENS Lyon, France
†University of Basel, Switzerland
‡Georgia Institute of Technology, Atlanta, GA, USA
§University of Tennessee, Knoxville, TN, USA

***Abstract*—This paper combines checkpointing and replication for the reliable execution of linear workflows. While both methods have been studied separately, their combination has not yet been investigated despite its promising potential to minimize the execution time of linear workflows in failure-prone environments. The combination raises new problems: for each task, we have to decide whether to checkpoint and/or replicate it. We provide an optimal dynamic programming algorithm of quadratic complexity to solve both problems. This dynamic programming algorithm has been validated through extensive simulations that reveal the conditions in which checkpointing only, replication only, or the combination of both techniques lead to improved performance.**

I. Introduction

Several high-performance computing (HPC) applications are designed as a succession of (typically large) tightly-coupled computational kernels, or tasks, that should be executed in sequence [6], [11], [22]. These parallel tasks are executed on the whole platform, and they exchange data at the end of their execution. In other words, the task graph is a linear chain, and each task (except maybe the first one and the last one) reads data from its predecessor and produces data for its successor. Such linear chains of tasks also appear in image processing applications [26], and are usually called *linear workflows* [35].

The first objective when dealing with linear workflows is to ensure an *efficient execution*, which amounts to minimizing the total parallel execution time, or makespan. However, a *reliable execution* is also critical to performance. Indeed, large-scale platforms are increasingly subject to failures. Scale is the enemy here: even if each computing resource is very reliable, with, say, a Mean Time Between Failures (MTBF) of 10 years, meaning that each resource will experience a failure only every 10 years on average, a platform composed of one million of such resources will experience a failure every five minutes [21]. Hence, fault-tolerance techniques to mitigate the impact of failures are required to ensure a correct execution of the application [24]. The standard approach is checkpoint, rollback and recovery [9], [15]: in the context of linear workflow applications, each task can decide to take a checkpoint after it has been correctly executed. A checkpoint is simply a file including all intermediate results and associated data that is saved on a storage medium resilient to failures; it can be either the memory of another processor, a local disk or a remote disk. This file can be recovered if a successor task experiences a failure later on in the execution. If there is an error while some task is executing, the application has to roll back to the last checkpointed task (or to start again from scratch if no checkpoint was taken). Then the checkpoint is read from the storage medium (recovery phase), and execution resumes from that task onward. If the checkpoint was taken many tasks before a failure strikes, there is a lot of re-execution involved, which calls for frequent checkpoints. However, checkpointing incurs a significant overhead, and is a mere waste of resources if no failure strikes. Altogether, there is a trade-off to be found, and one may want to checkpoint only carefully selected tasks.

Another approach to address failures consists in replicating the work: we can for instance execute a task twice, in parallel, using only half of the platform for each replica, in order to maximize the chance of success. Indeed, if one of the executions succeeds without failure, we can keep going to the next task. Even though this approach has a high cost in terms of computing resources (half of the platform is *wasted* if no failure strikes), several authors have recently advocated the use of replication in HPC in the recent years [30], [42], [17], [19]. Indeed, if there are too many failures, an application using only checkpointing may experience too many recoveries and re-execution delays in order to progress efficiently. Furthermore, parallel tasks are often following *Amdahl's law* [1], i.e., they include a sequential part that will take the same time, whatever the number of processors allocated to the task. Hence, using twice more processors to execute a task does not mean that the execution will be twice faster. Coupling a better failure-free efficiency with a better resilience to failures makes duplication worth investigating for linear workflows[1].

While both checkpointing and replication have been extensively studied separately, their combination has not yet been investigated despite its promising potential to minimize the execution time in failure-prone environments, in particular in the context of linear workflows. The contributions of this work are the following:

- We provide a detailed model for the reliable execution of linear workflows, where each task can be replicated or not, and where the checkpoint cost depends both on the number of processors executing the task, and on whether the task is replicated or not;
- We design an optimal dynamic programming algorithm that minimizes the makespan of a linear workflow with

[1]We only consider *duplication* in this work. Having three replicas (*triplication*) is possible but useful only with extremely high failure rates that cannot be mitigated via duplication, which are unlikely in HPC systems [19].

n tasks, with a quadratic complexity, in the presence of fail-stop errors;
- We conduct extensive experiments to evaluate the impact of using both replication and checkpointing during execution, and compare them to an execution without replication;
- We provide guidelines about when it is beneficial to employ checkpointing only, replication only, or to combine both techniques together.

The paper is organized as follows. Section II details the model and formalizes the objective function and the optimization problem. Section III presents a preliminary result for the dynamic programming algorithm: we explain how to compute the expected time needed to execute a single task (replicated or not), assuming that its predecessor has been checkpointed. The dynamic programming algorithm is outlined in Section IV and the experimental validation is provided in Section V. Finally, related work is discussed in Section VI, and we conclude in Section VII.

II. Model and objective

This section details the framework of this study. We start with the application and platform models, then detail checkpointing and replication, and finally state the optimization problem.

A. Application model

We target applications whose workflow represents a linear chain of parallel tasks. More precisely, we have a chain $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_n$ of n parallel tasks T_i, $1 \leq i \leq n$. Hence, T_1 must be completed before executing T_2, and so on.

Here, each T_i is a parallel task whose speedup profile obeys *Amdahl's law* [1]: the total work is w_i, with a sequential fraction $\alpha_i w_i$ and the remaining fraction $(1-\alpha_i)w_i$ perfectly parallel. The (failure-free) execution time of T_i using q_i processors is thus $w_i\left(\alpha_i + \frac{1-\alpha_i}{q_i}\right)$. Without loss of generality, we assume that processors execute the tasks at unit speed, and we speak of time units and work units interchangeably. While our study is agnostic of task granularity, it applies primarily to frameworks where tasks represent large computational entities whose execution takes, say, from a few minutes up to tens of minutes. In such frameworks, it may be worthwhile to replicate or checkpoint the task to mitigate the impact of failures.

B. Execution platform

We target a *homogeneous* platform with p processors P_i, $1 \leq i \leq p$. We assume that the platform is subject to fail-stop errors whose inter-arrival times follow an Exponential distribution. More precisely, let λ_{ind} be the error rate of each individual processor P_i: the probability of having a fail-stop error striking P_i within T time-units is $\mathbb{P}(X \leq T) = 1 - e^{-\lambda_{ind}T}$. Then, a computation on $q \leq p$ processors has an error rate $q\lambda_{ind}$, and the probability of having a fail-stop error within T time-units becomes $1 - e^{-q\lambda_{ind}T}$ [21].

C. Checkpointing

The output of each task T_i can be checkpointed in time C_i. When an error strikes, we first incur a downtime D, and then we must start the execution from the task following the last checkpoint. Hence, if T_j is the last checkpointed task, the execution starts again at task T_{j+1}, and the recovery cost is R_{j+1}, which amounts to reading the checkpoint of task T_j. The checkpoint cost C_i and recovery cost R_i clearly depend upon the checkpoint protocol and storage medium, as well as upon the number q_i of enrolled processors. In this work, we adopt a quite general formula for checkpoint times and use

$$C_i(q_i) = a_i + \frac{b_i}{q_i} + c_i q_i \tag{1}$$

to model the time to save a checkpoint after T_i executed with q_i processors. Here, $a_i + \frac{b_i}{q_i}$ represents the I/O overhead to write the task output file M_i to the storage medium. For in-memory checkpointing [41], $a_i + \frac{b_i}{q_i}$ is the communication time with latency a_i; then we have $\frac{b_i}{q_i} = \frac{M_i}{\tau_{net} q_i}$, where τ_{net} is the network bandwidth (each processor stores $\frac{M_i}{q_i}$ data items). For coordinated checkpointing to stable storage, there are two cases: if the storage system's bandwidth is the I/O bottleneck, then $a_i = \beta + \frac{M_i}{\tau_{io}}$ and $b_i = 0$, where β is a start-up time and τ_{io} is the I/O bandwidth; otherwise, if the network is the I/O bottleneck, we retrieve the same formula as for in-memory checkpointing. Finally, $c_i q_i$ represents the message passing overhead that grows linearly with the number of processors, in order for all processors to reach a global consistent state [15], [43].

For the cost of recovery, we assume a similar formula:

$$R_i(q_i) = a'_i + \frac{b'_i}{q_i} + c'_i q_i. \tag{2}$$

If we further assume that reading and writing from/to the storage medium have same cost, we have $R_{i+1}(q_i) = C_i(q_i)$ for $1 \leq i \leq n-1$, since recovering for task T_{i+1} amounts to reading the checkpoint from task T_i.

Finally, we assume that there is a fictitious task T_0 of zero weight ($w_0 = 0$) that is always checkpointed, so that $R_1(q_1)$ represents the time for I/O input from the external world. Similarly, we systematically checkpoint the last task T_n, in order to account for the I/O output time $C_n(q_n)$.

D. Replication

When executing a task, we envision two possibilities: either the task is not replicated, or it is replicated. Consider a task T_i, and assume for simplicity that the predecessor T_{i-1} of T_i has been checkpointed. If it is not the case, i.e., if the predecessor T_{i-1} of T_i is not checkpointed, we have to roll back to the last checkpointed task, say T_k where $k < i-1$, whenever a failure strikes, and re-execute the whole segment from T_{k+1} to T_i instead of just T_i.

Without replication, a single copy of T_i is executed on the whole platform, hence with $q_i = p$ processors. Then we let $\mathbb{E}^{norep}(i)$ denote the expected execution time of T_i when accounting for failures. We attempt a first execution, which takes $w_i\left(\alpha_i + \frac{1-\alpha_i}{p}\right)$ if no failure strikes. But if some failure

does strike, we must account for the time that has been lost (between the beginning of the execution and the failure), then perform a downtime D, a recovery $R_i(p)$ (since we use the whole platform for T_i), and then re-execute T_i from scratch. Similarly, if we decide to checkpoint after T_i, we need $C_i(p)$ time units. We explain how to compute $\mathbb{E}^{norep}(i)$ in Section III.

With replication, two copies of T_i are executed in parallel, each with $q_i = \frac{p}{2}$ processors. If no failure strikes, both copies finish execution in time $w_i\left(\alpha_i + \frac{1-\alpha_i}{\frac{p}{2}}\right)$, since each copy uses $\frac{p}{2}$ processors. If a failure strikes one copy, we proceed as before, account for the downtime D, recover (in time $R_i(\frac{p}{2})$ now), and restart execution of that copy. Then there are two cases: (i) if the second copy successfully completes its first execution, the failure has no impact and the execution time remains the same as the failure-free execution time; (ii) however, if the second copy also fails to execute, we resume its execution, and iterate until one copy successfully completes. Of course, case (ii) is less likely to happen than case (i), which explains why replication can be useful. Finally, if we decide to checkpoint after T_i, the first successful copy will take the checkpoint in time $C_i(\frac{p}{2})$.

Replication raises several complications in terms of checkpoint and recovery costs. When a replicated task T_i is checkpointed, we can enforce that only one copy (the first one to complete execution) would write the output data onto the storage medium, hence with a cost $C_i(\frac{p}{2})$, as stated above. Similarly, when a single copy of a replicated task T_i performs a recovery after a failure, the cost would be $R_i(\frac{p}{2})$. However, in the unlikely event where both copies are struck by a failure at close time instances, their recoveries would overlap, and the cost can vary anywhere between $R_i(\frac{p}{2})$ and $2R_i(\frac{p}{2})$, depending upon the amount of contention, the length of the overlap and where the I/O bottleneck lies. We will experimentally evaluate the impact of the recovery cost with replication in Section V. For simplicity, in the rest of the paper, we use C_i^{rep} for the checkpoint cost of T_i when it is replicated, and C_i^{norep} when it is not. Similarly, we use R_i^{rep} for the recovery cost when T_i is replicated, and R_i^{norep} when it is not. Note that the recovery cost of T_i depends upon whether it is replicated or not, but does not depend upon whether the checkpointed task T_{i-1} was replicated or not, since we need to read the same file from the storage medium in both cases. The values of C_i^{rep} and C_i^{norep} can be instantiated from Equation (1) and those of R_i^{rep} and R_i^{norep} can be instantiated from Equation (2).

Finally, we let $\mathbb{E}^{rep}(i)$ denote the expected execution time of T_i with replication and when accounting for failures, when T_{i-1} is checkpointed. The derivation of $\mathbb{E}^{rep}(i)$ is much more complicated than for $\mathbb{E}^{norep}(i)$ and represents a new contribution of this work. We explain how to compute $\mathbb{E}^{rep}(i)$ in Section III-B.

E. Optimization problem

The objective is to *minimize the expected makespan* of the workflow in the presence of fail-stop errors. For each task, we have four choices: either we replicate the task or not, and either we checkpoint it or not. We point out that none of these decisions can be made locally. Instead, we need to account for previous decisions and optimize globally. Our major contribution of this work is to provide an optimal dynamic programming algorithm to solve this problem, which we denote as CHAINSREPCKPT.

We point out that CHAINSCKPT, the simpler problem without replication, i.e., optimally placing checkpoints for a chain of tasks, has been extensively studied. The first dynamic programming algorithm to solve CHAINSCKPT appears in the pioneering paper of Toueg and Babaoğlu [36] back in 1984 (see Section VI on related work for further references). Adding replication dramatically complicates the solution. Here is an intuitive explanation: When the algorithm recursively considers a segment of tasks from T_i to T_j, where T_{i-1} and T_j are both checkpointed and no intermediate task T_k, $i \leq k < j$ is checkpointed, there are many cases to consider to account for possible different values in: (i) execution time, since some tasks in the segment may be replicated; (ii) checkpoint, whose cost depends upon whether T_j is replicated or not; and (iii) recovery, whose cost depends upon whether T_i is replicated or not. We provide all details in Section IV.

III. COMPUTING $\mathbb{E}^{norep}(i)$ AND $\mathbb{E}^{rep}(i)$

This section details how to compute the expected time needed to execute a task T_i, assuming that the predecessor of T_i has been checkpointed. Hence, we need to re-execute only T_i when a failure strikes. We explain how to deal with the general case of re-executing a segment of tasks, some of them replicated, in Section IV. We start with the case where T_i is not replicated. It is already known how to compute $\mathbb{E}^{norep}(i)$ [21], but we present this case to help the reader follow the derivation in Section III-B for the case where T_i is replicated, which is new and much more involved.

A. Computing $\mathbb{E}^{norep}(i)$

To compute $\mathbb{E}^{norep}(i)$, the average execution time of T_i with p processors without replication, we conduct a case analysis:

- Either an error strikes during the execution, and in this case we lose some work and then need to re-execute the task;
- Either there is no error, and in this case we only need the failure-free execution time $T_i^{norep} = w_i\left(\alpha_i + \frac{1-\alpha_i}{p}\right)$.

This leads to the following recursive formula:

$$\mathbb{E}^{norep}(i) = \mathbb{P}(X_p \leq T_i^{norep})\Big(T_{lost}^{norep}(T_i^{norep}) + D + R_i^{norep} + \mathbb{E}^{norep}(i)\Big) + (1 - \mathbb{P}(X_p \leq T_i^{norep}))T_i^{norep}, \quad (3)$$

where $\mathbb{P}(X_p \leq t)$ is the probability of having a failure on one of the p processors before time t, i.e., $\mathbb{P}(X_p \leq t) = 1 - e^{-\lambda_{ind} p t}$. The time lost when a failure strikes is the expectation of the random variable X_p, knowing that the error stroke before the end of the task. We compute it as follows:

$$T_{lost}^{norep}(T_i^{norep}) = \int_0^{+\infty} x\mathbb{P}(X_p = x | X_p \leq T_i^{norep})dx$$
$$= \frac{1}{\mathbb{P}(X_p \leq T_i^{norep})}\int_0^{T_i^{norep}} x\mathbb{P}(X_p = x)dx = \frac{1}{\mathbb{P}(X_p \leq T_i^{norep})}\int_0^{T_i^{norep}} x\frac{d\mathbb{P}(X_p \leq x)}{dx}dx$$

After integration, we get the formula:

$$T_{lost}^{norep}(T_i^{norep}) = \frac{1}{\lambda_{ind}p} - \frac{t}{e^{\lambda_{ind}pT_i^{norep}} - 1}. \quad (4)$$

Replacing the terms in Equation (3) and solving, we derive:

$$\mathbb{E}^{norep}(i) = (e^{\lambda_{ind}pT_i^{norep}} - 1)(\frac{1}{\lambda_{ind}p} + D + R_i^{norep}). \quad (5)$$

Recall that $T_i^{norep} = w_i\left(\alpha_i + \frac{1-\alpha_i}{p}\right)$ in Equation (5). Finally, if we decide to checkpoint T_i, we simply add C_i^{norep} to $\mathbb{E}^{norep}(i)$.

B. Computing $\mathbb{E}^{rep}(i)$

We now discuss the case where T_i is replicated; each copy executes with $\frac{p}{2}$ processors. To compute $\mathbb{E}^{rep}(i)$, the expected execution time of T_i with replication, we conduct the same case analysis:

- Either two failures strike before the end of the task, with one failure striking each copy, and we have lost some work and need to re-execute the task;
- Either (at least) one copy is not hit by any failure, and in this case we only need the failure-free execution time $T_i^{rep} = w_i\left(\alpha_i + \frac{1-\alpha_i}{\frac{p}{2}}\right)$.

This leads to the following formula:

$$\begin{aligned}\mathbb{E}^{rep}(i) = \mathbb{P}(Y_p \leq T_i^{rep})\Big(T_{lost}^{rep}(T_i^{rep}) + D + R_i^{rep} + \mathbb{E}^{rep}(i)\Big)\\ +(1 - \mathbb{P}(Y_p \leq T_i^{rep}))T_i^{rep}\end{aligned} \quad (6)$$

where $\mathbb{P}(Y_p \leq t)$ is the probability of having a failure on both replicas of $\frac{p}{2}$ processors before time t, i.e., $\mathbb{P}(Y_p \leq t) = (1 - e^{-\frac{\lambda_{ind}p}{2}t})^2$. The time lost when both copies fail can be computed in a similar way as before:

$$T_{lost}^{rep}(T_i^{rep}) = \frac{1}{\mathbb{P}(Y_p \leq T_i^{rep})} \int_0^{T_i^{rep}} x\frac{d\mathbb{P}(Y_p \leq x)}{dx}dx.$$

After computation and verification using a Maple sheet, we obtain the following result:

$$T_{lost}^{rep}(T_i^{rep}) = \frac{(-2\lambda_{ind}pT_i^{rep} - 4)e^{-\frac{\lambda_{ind}pT_i^{rep}}{2}} + (\lambda_{ind}pT_i^{rep} + 1)e^{-\lambda_{ind}pT_i^{rep}} + 3}{(e^{-\frac{\lambda_{ind}pT_i^{rep}}{2}} - 1)^2\lambda_{ind}p}. \quad (7)$$

Replacing the terms in Equation (6) and solving, we get:

$$\begin{aligned}\mathbb{E}^{rep}(i) = \frac{3e^{\lambda_{ind}pT_i^{rep}} - 4e^{\frac{\lambda_{ind}p}{2}T_i^{rep}} + 1}{2e^{\frac{\lambda_{ind}p}{2}T_i^{rep}} - 1}\frac{1}{\lambda_{ind}p}\\ + (\frac{e^{\lambda_{ind}pT_i^{rep}}}{2e^{\frac{\lambda_{ind}p}{2}T_i^{rep}} - 1} - 1)(D + R_i^{rep}).\end{aligned} \quad (8)$$

Recall that $T_i^{rep} = w_i\left(\alpha_i + \frac{1-\alpha_i}{\frac{p}{2}}\right)$ in Equation (8). Finally, if we decide to checkpoint T_i, we simply add C_i^{rep} to $\mathbb{E}^{rep}(i)$.

IV. Optimal DP algorithm

In this section, we provide a dynamic programming algorithm to solve the ChainsRepCkpt problem for a linear chain of n tasks.

Theorem 1. *The optimal solution to the* ChainsRepCkpt *problem can be obtained using a dynamic programming algorithm in $O(n^2)$ time, where n is the number of tasks in the chain.*

Proof. The algorithm recursively computes the expectation of the optimal time required to execute tasks T_1 to T_i and then checkpoint T_i. As already mentioned, we need to distinguish two cases, according to whether T_i is replicated or not, because the cost of the final checkpoint depends upon this decision. Hence, we recursively compute two different functions:

- $T_{opt}^{rep}(i)$, the expectation of the optimal time required to execute tasks T_1 to T_i, knowing that T_i is replicated;
- $T_{opt}^{norep}(i)$, the expectation of the optimal time required to execute tasks T_1 to T_i, knowing that T_i is not replicated.

Note that checkpoint time is not included in $T_{opt}^{rep}(i)$ nor $T_{opt}^{norep}(i)$. The solution to ChainsRepCkpt will be given by

$$\min\left\{T_{opt}^{rep}(n) + C_n^{rep}, T_{opt}^{norep}(n) + C_n^{norep}\right\}. \quad (9)$$

We start with the computation of $T_{opt}^{rep}(j)$ for $1 \leq j \leq n$, hence assuming that the last task T_j is replicated. We express $T_{opt}^{rep}(j)$ recursively as follows:

$$T_{opt}^{rep}(j) = \min_{1\leq i<j}\left\{\begin{array}{l} T_{opt}^{rep}(i) + C_i^{rep} + T_{NC}^{rep,rep}(i+1,j),\\ T_{opt}^{rep}(i) + C_i^{rep} + T_{NC}^{norep,rep}(i+1,j),\\ T_{opt}^{norep}(i) + C_i^{norep} + T_{NC}^{rep,rep}(i+1,j),\\ T_{opt}^{norep}(i) + C_i^{norep} + T_{NC}^{norep,rep}(i+1,j),\\ R_1^{rep} + T_{NC}^{rep,rep}(1,j),\\ R_1^{norep} + T_{NC}^{norep,rep}(1,j)\end{array}\right\} \quad (10)$$

In Equation (10), T_i corresponds to the last checkpointed task before T_j, and we try all possible locations T_i for taking a checkpoint before T_j. The first four lines correspond to the case where there is indeed an intermediate task T_i that is checkpointed, while the last two lines correspond to the case where no checkpoint at all is taken until after T_j.

The first two lines of Equation (10) apply to the case where T_i is replicated. Line 1 is for the case when T_{i+1} is replicated, and line 2 when it is not. In the first line of Equation (10), $T_{NC}^{rep,rep}(i+1,j)$ denotes the optimal time to execute tasks T_{i+1} to T_j without any intermediate checkpoint, knowing that T_i is checkpointed, and both T_{i+1} and T_j are replicated. If T_{i+1} is not replicated, we use the second line of Equation (10), where $T_{NC}^{norep,rep}(i+1,j)$ is the counterpart of $T_{NC}^{rep,rep}(i+1,j)$, except that it assumes that T_{i+1} is not replicated. This information on T_{i+1} (replicated or not) is needed to compute the recovery cost when executing tasks T_{i+1} to T_j and experimenting a failure.

Lines 3 and 4 apply to the case where T_i is not replicated, with similar notations as before. In the first four lines, no task between T_{i+1} and T_{j-1} is checkpointed, hence the notation NC for no checkpoint.

If no checkpoint at all is taken before T_j (this corresponds to the case $i = 0$), we use the last two lines of Equation (10): we include the cost to read the initial input, which depends whether T_1 is replicated (in line 5) or not (in line 6).

We have a very similar equation to express $T_{opt}^{norep}(j)$ recursively, with obvious notations:

$$T_{opt}^{norep}(j) = \min_{1 \le i < j} \left\{ \begin{array}{l} T_{opt}^{rep}(i) + C_i^{rep} + T_{NC}^{rep,norep}(i+1,j), \\ T_{opt}^{rep}(i) + C_i^{rep} + T_{NC}^{norep,norep}(i+1,j), \\ T_{opt}^{norep}(i) + C_i^{norep} + T_{NC}^{rep,norep}(i+1,j), \\ T_{opt}^{norep}(i) + C_i^{norep} + T_{NC}^{norep,norep}(i+1,j), \\ R_1^{rep} + T_{NC}^{rep,norep}(1,j), \\ R_1^{norep} + T_{NC}^{norep,norep}(1,j) \end{array} \right\} \tag{11}$$

To synthesize notations, we have defined $T_{NC}^{A,B}(i+1,j)$, with $A, B \in \{rep, norep\}$, as the optimal time to execute tasks T_{i+1} to T_j without any intermediate checkpoint, knowing that T_i is checkpointed, T_{i+1} is replicated if and only if $A = rep$, and T_j is replicated if and only if $B = rep$. In a nutshell, we have to account for the possible replication of the first task T_{i+1} after the last checkpoint, and of the last task T_j, hence the four cases.

There remains to compute $T_{NC}^{A,B}(i,j)$ for all $1 \le i, j \le n$ and $A, B \in \{rep, norep\}$. This is still not easy, because there remains to decide which intermediate tasks should be replicated. In addition to the status of T_j (replicated or not, according to the value of B), the only thing we know so far is that the only checkpoint that we can recover from while executing tasks T_i to T_j is the checkpoint taken after task T_{i-1}, hence we need to re-execute from T_i whenever a failure strikes. Furthermore, T_i is replicated if and only if $A = rep$, hence we know the corresponding cost for recovery, R_i^A. Letting $T_{NC}^{A,B}(i,j) = 0$ whenever $i > j$, we can express $T_{NC}^{A,B}(i,j)$ for $1 \le i \le j \le n$ as follows:

$$\begin{aligned} T_{NC}^{A,B}(i,j) = \quad & \min \left\{ T_{NC}^{A,rep}(i,j-1), T_{NC}^{A,norep}(i,j-1) \right\} \\ & + T^{A,B}(j \mid i). \end{aligned}$$

Here the new (and final) notation $T^{A,B}(j \mid i)$ is simply the time needed to execute task T_j, knowing that a failure during T_j implies to recover from T_i. Indeed, to execute tasks T_i to T_j, we account recursively for the time to execute T_i to T_{j-1}; T_{i-1} is still checkpointed; T_i is replicated if and only if $A = rep$, T_j is replicated if and only if $B = rep$, and we consider both cases whether T_{j-1} is replicated or not. The time lost in case of a failure during T_j depends whether T_j is replicated or not, and we need to restart from T_i in case of failure, hence the notation $T^{A,B}(j \mid i)$, representing the expected execution time for task T_j with or without replication (depending on B), given that we need to restart from T_i if there is a failure (and T_i is replicated if and only if $A = rep$).

The last step is hence to express these execution times. We start with the case where T_j is not replicated:

$$\begin{aligned} T^{A,norep}(j \mid i) = & \left(1 - e^{-\lambda T_j^{norep}}\right) \left(T_{lost}^{norep}(T_j^{norep}) + D + R_i^A \right. \\ & \left. + \min \left\{ T_{NC}^{A,rep}(i,j-1), T_{NC}^{A,norep}(i,j-1) \right\} + T^{A,norep}(j \mid i) \right) \\ & + e^{-\lambda T_j^{norep}} \left(T_j^{norep}\right). \end{aligned}$$

The term in $e^{-\lambda T_j^{norep}}$ represents the case without failure, where the execution time is simply T_j^{norep}. When a failure strikes, we account for $T_{lost}^{norep}(T_j^{norep})$, the time lost within T_j, and whose value is given by Equation (4). Then we pay a downtime and a recovery (with a cost depending on A). Next, we need to re-execute all the tasks since the last checkpoint (T_i to T_{j-1}) and take the minimal value obtained out of the execution where T_{j-1} is replicated or not; finally we execute T_j again (with a time $T^{A,norep}(j \mid i)$).

The formula is similar with replication, where the probability of failure accounts for the fact that we need to recover only if both replicas fail:

$$\begin{aligned} T^{A,rep}(j \mid i) = & \left(1 - e^{-\frac{\lambda T_j^{rep}}{2}}\right)^2 \left(T_{lost}^{rep}(T_j^{rep}) + D + R_i^A \right. \\ & \left. + \min \left\{ T_{NC}^{A,rep}(i,j-1), T_{NC}^{A,norep}(i,j-1) \right\} + T^{A,rep}(j \mid i) \right) \\ & + \left(1 - \left(1 - e^{-\frac{\lambda T_j^{rep}}{2}}\right)^2\right) \left(T_j^{rep}\right). \end{aligned}$$

Note that the value of $T_{lost}^{rep}(T_j^{rep})$ is given by Equation (7). Overall, we need to compute the $O(n^2)$ intermediate values $T^{A,B}(j \mid i)$ and $T_{NC}^{A,B}(i,j)$ for $1 \le i, j \le n$ and $A, B \in \{rep, norep\}$, and each of these take constant time. There are $O(n)$ values $T_{opt}^{A}(i)$, for $1 \le i \le n$ and $A \in \{rep, norep\}$, and these perform a minimum over at most $6n$ elements, hence they can be computed in $O(n)$. The overall complexity is therefore $O(n^2)$. □

V. Experiments

In this section, we evaluate the advantages of adding replication to checkpointing. We first describe the evaluation framework in Section V-A, then we compare *checkpoint with replication* to *checkpoint only* in Section V-B. In Section V-C, we assess the impact of the different model parameters on the performance of the optimal strategy. Finally, Section V-D compares the performance of the optimal solution to alternative sub-optimal solutions[2].

A. Experimental setup

We fix the total work in the chain to $W = 10,000$ seconds, and we rely on five different work distributions, where all tasks are fully parallel ($\alpha_i = 0$):

- Uniform: every task is of length $\frac{W}{n}$, i.e., identical tasks.
- Increasing: the length of the tasks constantly increases, i.e., task T_i has length $i\frac{2W}{n(n+1)}$.
- Decreasing: the length of the tasks constantly decreases, i.e., task T_i has length $(n-i+1)\frac{2W}{n(n+1)}$.
- HighLow: the chain is formed by big tasks followed by small tasks. The large tasks represent 60% of the total work and there are $\lceil \frac{n}{10} \rceil$ such tasks. Small tasks represent the remaining 40% of the total work and consequently there are $n - \lceil \frac{n}{10} \rceil$ small tasks.
- Random: task lengths are chosen uniformly at random between $\frac{W}{2n}$ and $\frac{3W}{2n}$. If the total work of the first i tasks reaches W, the weight of each task is multiplied by $\frac{i}{n}$ so that we can continue adding the remaining tasks.

Experiments with increasing sequential part (α_i) for the tasks are available in the companion research report [3]. Setting

[2]The simulator is publicly available at http://graal.ens-lyon.fr/~yrobert/chainsrep.zip so that interested readers can instantiate their preferred scenarios and repeat the same simulations for reproducibility purpose.

$\alpha_i = 0$ amounts to being in the worse possible case for replication, since the tasks will fully benefit of having twice as much processors when not replicated.

For simplicity, we assume that checkpointing costs are equal to the corresponding recovery costs, assuming that read and write operations take approximately the same amount of time, i.e., $R_{i+1}^{norep} = C_i^{norep}$. For replicated tasks, we set $C_i^{rep} = \alpha C_i^{norep}$ and $R_i^{rep} = \alpha R_i^{norep}$, where $1 \leq \alpha \leq 2$, and we assess the impact of parameter α in Section V-C. In the following experiments, we measure the performance of a solution by evaluating the associated normalized expected makespan, i.e., the expected execution time needed to compute all the tasks in the chain, with respect to the execution time without errors, checkpoints, or replicas.

B. Comparison to checkpoint only

We start with an analysis of the solutions obtained by running the optimal dynamic programming (DP) algorithm CHAINSREPCKPT on chains of 20 tasks for the five different work distributions described in Section V-A. We also run a variant of CHAINSREPCKPT that does not perform any replication, hence using a simplified DP algorithm, that is called CHAINSCKPT.

We vary the error rate $\lambda_{ind}p$ from 10^{-8} to 10^{-2}. Note that when $\lambda_{ind}p = 10^{-3}$, we expect an average of 10 errors per execution of the entire chain (neglecting potential errors during checkpoints and recoveries). The checkpoint cost $C_i^{norep} = a_i$ is constant per task (hence $b_i = c_i = 0$) and varies from $10^{-3}T_i^{norep}$ to $10^{3}T_i^{norep}$. For replicated tasks, we set $\alpha = 1$ in this experiment, i.e., $C_i^{rep} = C_i^{norep}$ and $R_i^{rep} = R_i^{norep}$.

Figure 1 presents the results of these experiments for the UNIFORM distribution. We are interested in the number of checkpoints and replicas in the optimal solution: *None* means that no task is checkpointed nor replicated, *Checkpointing Only* means that some tasks are checkpointed but no task is replicated, *Replication Only* means that some tasks are replicated, but no task is checkpointed, and *Checkpointing+Replication* means that some tasks are checkpointed and some tasks are replicated. First, we observe that when the checkpointing cost is less than or equal to the length of a task (on the left of the black line), the optimal solution does not use replication. However, if the checkpointing cost exceeds the length of one task (on the right of the black vertical bar), replication proves useful in some cases. In particular, when the error rate $\lambda_{ind}p$ is medium to high (i.e., 10^{-6} to 10^{-4}), we note that only replication is used, meaning that no checkpoint is taken and that replication alone is a better strategy to prevent any error from stopping the application. When the error rate is the highest (i.e., 10^{-4} or higher), replication is added to the checkpointing strategy to ensure maximum reliability. It may seem unusual to use replication alone when checkpointing costs increase. This is because the recovery cost has to be taken into account as well, in addition to re-executing the tasks that have failed. Replication is added to reduce this risk: if successful, there is no recovery cost to pay for, nor any task to re-execute. Finally, note that for small error rates and checkpointing costs, only checkpoints are used, because their cost is smaller than the

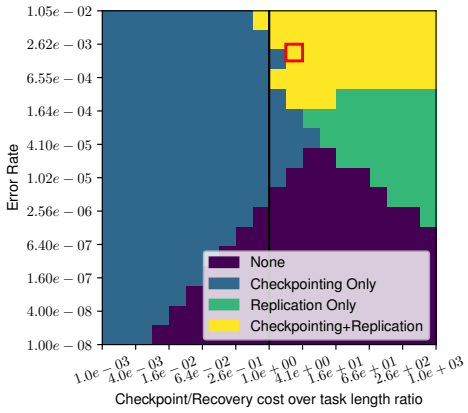


Figure 1: Impact of checkpoint/recovery cost and error rate on the usage of checkpointing and replication. Total work is fixed to $10,000$s and is distributed uniformly among $n = 20$ tasks (i.e., $T_1 = T_2 = \cdots = T_{20} = 500$s).

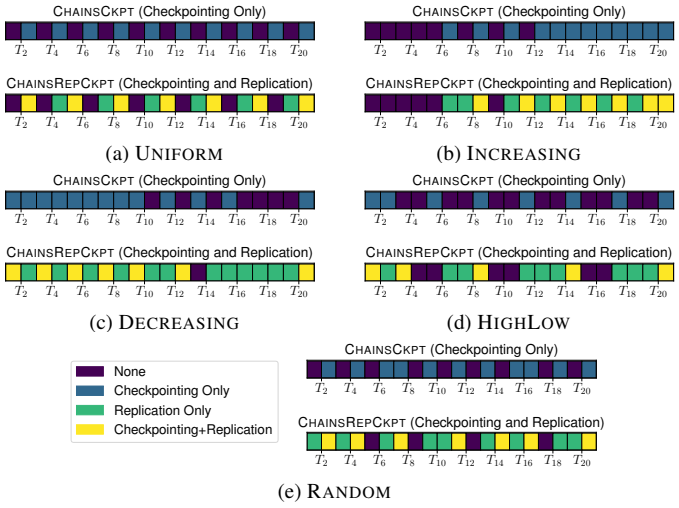


Figure 2: Optimal solutions obtained with the CHAINSCKPT algorithm (top) and the CHAINSREPCKPT algorithm (bottom) for the five work distributions.

average re-execution time in case of failure. We point out that similar results are obtained when using other work distributions (see the extended version [3]).

In the next experiment, we focus on scenarios where both checkpointing and replication are useful, i.e., we set the checkpointing cost to be twice the length of a task (i.e., $C_i^{norep} = a_i = 2T_i^{norep}$), and we set the error rate $\lambda_{ind}p$ to 10^{-3}, which corresponds to the case highlighted in red in Figure 1. Figure 2 presents the optimal solutions obtained with the CHAINSCKPT and CHAINSREPCKPT algorithms for the UNIFORM, INCREASING, DECREASING, HIGHLOW and RANDOM work distributions, respectively. First, for the UNIFORM work distribution, it is clear that the CHAINSREPCKPT strategy leads to a decrease of the number of checkpoints compared to the CHAINSCKPT strategy. Under the CHAINSCKPT strategy, a checkpoint is taken every two tasks, while under the CHAINSREPCKPT strategy, a checkpoint is taken every three tasks

instead, while two out of three tasks are also replicated. Then, for the INCREASING and DECREASING work distributions, the results show that most tasks should be replicated, while only the largest tasks are also checkpointed. A general rule of thumb is that replication only is preferred for small tasks while checkpointing and replication is reserved for larger tasks, where the probability of failure and the re-execution cost are the highest. Finally, we observe a similar trend for the HIGHLOW work distribution, where two of the first four large tasks are checkpointed and replicated.

Figure 3 compares the performance of CHAINSREPCKPT to the checkpoint-only strategy CHAINSCKPT. First, we observe that the expected normalized makespan of CHAINSCKPT remains almost constant at ≈ 4.5 for any number of tasks and for any work distribution. Indeed, in our scenario, checkpoints are expensive and the number of checkpoints that can be used is limited to ≈ 17 in the optimal solution, as shown in the middle plot. However, the CHAINSREPCKPT strategy can take advantage of the increasing number of smaller tasks by replicating them. In this scenario (high error rate and high checkpoint cost), this is clearly a winning strategy. The normalized expected makespan keeps decreasing as n increases, as the corresponding number of tasks that are replicated increases almost linearly. The CHAINSREPCKPT strategy reaches a normalized makespan of ≈ 2.6 for $n = 100$, i.e., a reduction of 35% compared to the normalized expected makespan of the CHAINSCKPT strategy. This is because replicated tasks tend to decrease the global probability of having a failure, thus reducing even more the number of checkpoints needed as seen previously. Regarding the HIGHLOW work distribution, we observe a higher optimal expected makespan for both the CHAINSCKPT and the CHAINSREPCKPT strategies. Indeed, in this scenario, the first tasks are very large (60% of the total work), which greatly increases the probability of failure and the associated re-execution cost.

C. Impact of error rate and checkpoint cost on the performance

Figure 4 shows the impact of three of the model parameters on the optimal expected normalized makespan of both CHAINSCKPT and CHAINSREPCKPT. First, we show the impact of the error rate $\lambda_{ind}p$ on the performance. The CHAINSREPCKPT algorithm improves the CHAINSCKPT strategy for large values of $\lambda_{ind}p$: replication starts to be used for $\lambda_{ind}p > 2.6 \times 10^{-4}$ and it reduces the makespan by $\approx 16\%$ for $\lambda_{ind}p = 10^{-3}$ and by up to $\approx 40\%$ when $\lambda_{ind}p = 10^{-2}$, where all tasks are checkpointed and replicated.

Then, we investigate the impact of the checkpointing cost with respect to the task length. As shown in Figure 1, replication is not needed for small checkpointing costs, i.e., when the checkpointing cost is between 0 and 0.8 times the cost of one task: in this scenario, all tasks are checkpointed and both strategies lead to the same makespan. When the checkpointing cost is between 0.9 and 1.6 times the cost of one task, CHAINSREPCKPT checkpoints and replicates half of the tasks. Overall, the CHAINSREPCKPT strategy improves the optimal normalized expected makespan by $\approx 11\%$ for a checkpointing cost ratio of 1.6, and by as much as $\approx 36\%$ when the checkpointing cost is five times the length of one task.

We now investigate the impact of the ratio between the checkpointing and recovery cost for replicated tasks and non-replicated tasks α and we present the results for $\alpha = 1$ ($C_i^{rep} = R_i^{rep} = C_i^{norep} = R_i^{norep}$), $\alpha = 1.5$ ($C_i^{rep} = R_i^{rep} = 1.5C_i^{norep} = 1.5R_i^{norep}$) and $\alpha = 2$ ($C_i^{rep} = R_i^{rep} = 2C_i^{norep} = 2R_i^{norep}$). As expected, the makespan increases with α, but it is interesting to note that the makespan converges towards a same lower-bound as the number of (smaller) tasks increases. As shown previously, when tasks are smaller, CHAINSREPCKPT favors replication over checkpointing, especially when the checkpointing cost is high, which means less checkpoints, recoveries and re-executions.

Finally, we evaluate the efficiency of both strategies when the number of processors increases. For this experiment, we instantiate the model using variable checkpointing costs, i.e., we do not use $b_i = c_i = 0$ anymore, so that the checkpointing/recovery cost depends on the number of processors. We set $n = 50$, $\lambda_{ind} = 10^{-7}$ and we make p vary from 10 to 10,000 (i.e., the global error rate varies between 10^{-6} and 10^{-3}). Figure 5 presents the results of the experiment using three different sets of values for a_i, b_i and c_i. We see that when b_i increases while c_i decreases, the replication becomes useless, even for the larger failure rate values. However, when the term $c_i p$ becomes large in front of $\frac{b_i}{p}$, we see that CHAINSREPCKPT is much better than CHAINSCKPT, as the checkpointing costs tend to decrease, in addition to all the other advantages investigated in the previous sections. With $p = 10,000$, the three different experiments show an improvement of 80.5%, 40.7% and 0% (from left to right, respectively).

D. Impact of the number of checkpoints and replicas

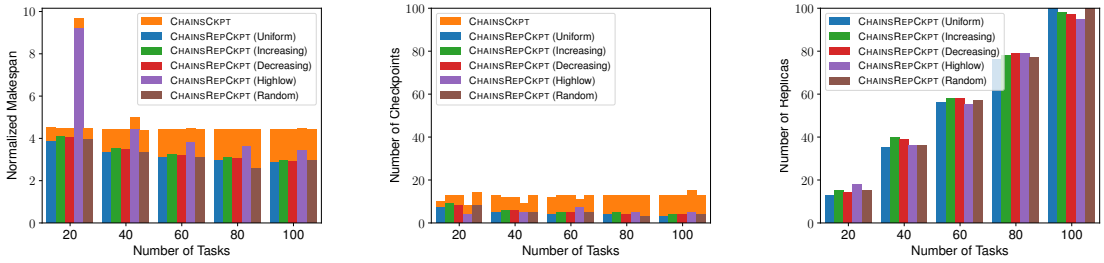
Figure 6 shows the impact of the number of checkpoints and replicas on the normalized expected makespan for different checkpointing costs and error rates $\lambda_{ind}p$ under the UNIFORM work distribution. We show that the optimal solution with CHAINSREPCKPT (highlighted in green) always matches the minimum value obtained in the simulations, i.e., the optimal number of checkpoints, number of replicas, and expected execution times are consistent. In addition, we show that in scenarios where both the checkpointing cost and the error rate are high, even a small deviation from the optimal solution can quickly lead to a large overhead.

VI. RELATED WORK

In this section, we discuss the work related to checkpointing and replication. Each of these mechanisms has been studied for coping with fail-stop errors and/or with silent errors. The present work combines checkpointing and replication for linear workflows in the presence of fail-stop errors.

A. Checkpointing

The de-facto general-purpose recovery technique in high-performance computing is checkpointing and rollback recovery [9], [16]. Checkpointing policies have been widely studied and we refer to [21] for a survey of various protocols.



(a) Uniform, Increasing, Decreasing, HighLow, Random distributions

Figure 3: Comparison of the ChainsCkpt and ChainsRepCkpt strategies for different numbers of tasks: impact on the makespan (left), number of checkpoints (middle) and number of replicas (right) with an error rate of $\lambda_{ind}p = 10^{-3}$ and a constant chekpointing/recovery cost $C_i^{norep} = C_i^{rep} = 1000$s.

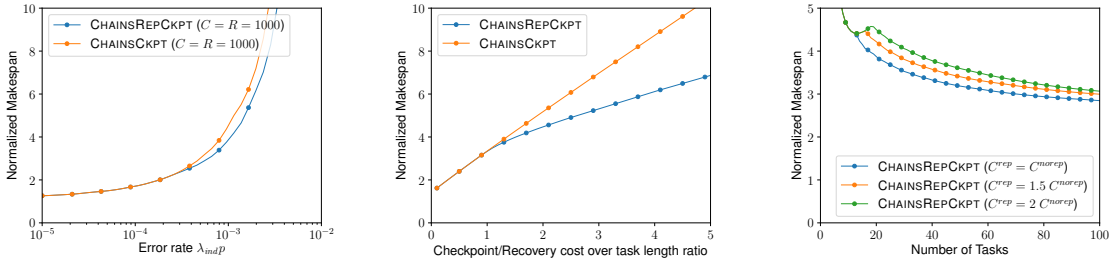
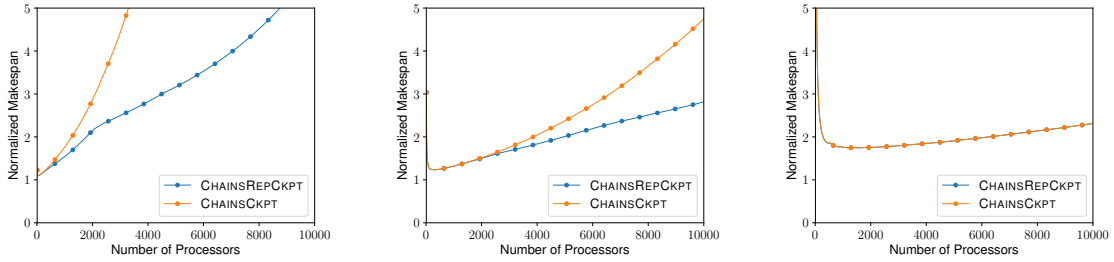


Figure 4: Impact of error rate $\lambda_{ind}p$ (left), checkpoint cost (middle) and ratio α between the checkpointing cost for replicated task C_i^{rep} over non-replicated tasks C_i^{norep} (right).



(a) $(100, 10000, 1)$ (b) $(100, 100000, 0.1)$ (c) $(100, 1000000, 0.01)$

Figure 5: Comparison of the ChainsCkpt and ChainsRepCkpt strategies for different numbers of processors, with different model parameter values for the checkpointing cost (a_i, b_i, c_i).

For divisible load applications where checkpoints can be inserted at any point in the execution for a nominal cost C, there exist well-known formulas proposed by Young [39] and Daly [12] to determine the optimal checkpointing period. For applications expressed as a linear workflow, as considered in the present work, the problem of finding the optimal checkpointing strategy, i.e., of determining which tasks to checkpoint, in order to minimize the expected execution time, has been solved by Toueg and Babaoğlu [36].

Single-level checkpointing schemes suffer from the intrinsic limitation that the cost of checkpointing and recovery grows with the failure probability, and becomes unsustainable at large scale [19], [5] (even with diskless or incremental checkpointing [28]). Recent advances in decreasing the cost of checkpointing include multi-level checkpointing approaches, or the use of SSD or NVRAM as secondary storage [7]. To reduce the I/O overhead, various two-level checkpointing protocols have been studied. Vaidya [37] proposed a two-level recovery scheme that tolerates a single node failure using a local checkpoint stored on a partner node. If more than one failure occurs during any local checkpointing interval, the scheme resorts to the global checkpoint. Silva and Silva [31] advocated for a similar scheme by using memory protected by XOR encoding to store local checkpoints. Di et al. [13] analyzed a two-level computational pattern, and proved that equal-length checkpointing segments constitute the optimal solution. Benoit et al. [4] relied on disk checkpoints to cope with fail-stop failures and used memory checkpoints coupled with error detectors to handle silent data corruptions. They derived first-order approximation formulas for the optimal pattern length as well as the number of memory checkpoints between two disk checkpoints. The present work employs single-level checkpointing (in memory or on stable storage) for individual tasks in linear workflows.

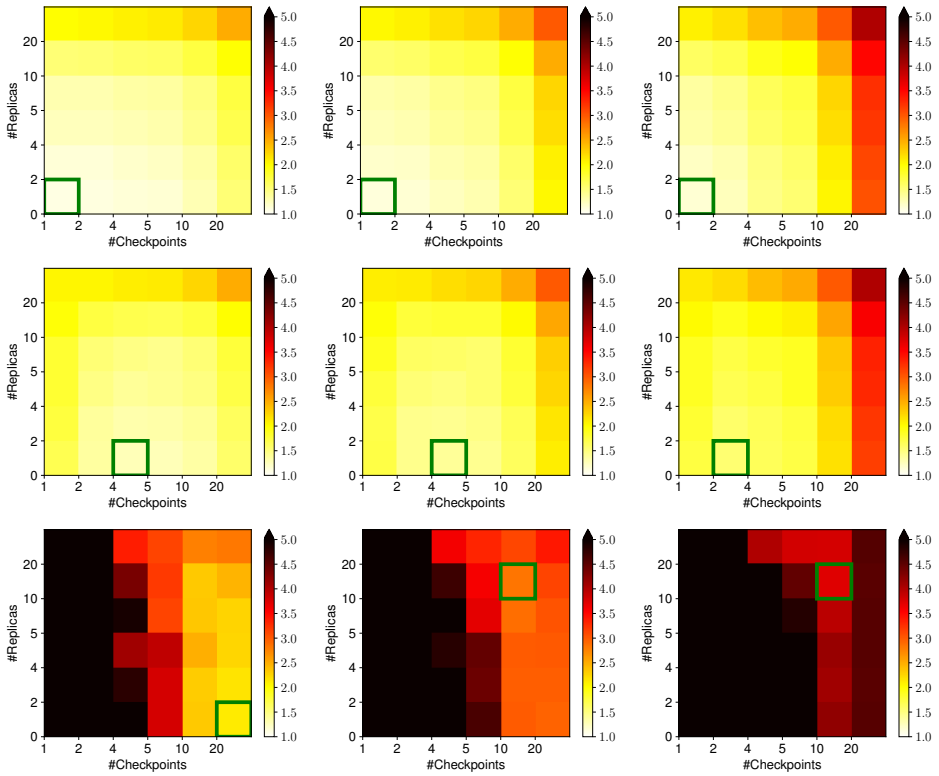


Figure 6: Impact of the number of checkpoints and replicas on the normalized expected makespan for $\lambda = 10^{-4}$ (top), $\lambda = 10^{-3}$ (middle) and $\lambda = 10^{-2}$ (bottom) and for checkpointing costs equal to $0.5 \times T_i^{norep}$ (left), $1 \times T_i^{norep}$ (middle) and $2 \times T_i^{norep}$ (right), with $C_i^{norep} = C_i^{rep}$ under UNIFORM work distribution. The optimal solution obtained with CHAINSREPCKPT always matches the minimum simulation value and is highlighted in green.

B. Replication

As mentioned earlier, this work only considers *duplication*. *Triplication* [25] (three replicas per task) is also possible yet only useful with extremely high failure rates, which are unlikely in HPC systems. The use of redundant MPI processes is analyzed in [8], [18], [19]. In particular, the work by Ferreira et al. [19] has studied the use of process replication for MPI applications, using two replicas per MPI process. They provide a theoretical analysis of parallel efficiency, an MPI implementation that supports transparent process replication (including failure detection, consistent message ordering among replicas, etc.), and a set of experimental and simulation results. Thread-level replication has been investigated in [40], [10], [29]. The present work targets selective task replication as opposed to full task replication in conjunction with selective task checkpointing to cope with fail-stop errors and minimize makespan.

Partial redundancy is studied in [14], [32], [33] (in combination with coordinated checkpointing) to decrease the overhead of full replication. Adaptive redundancy is introduced in [20], where a subset of processes is dynamically selected for replication. Earlier work [2] considered replication in the context of divisible load applications. Herein, task replication (including work and data) is studied in the context of linear workflows, which represent a harder case than that of divisible load applications as tasks cannot arbitrarily be divided and are executed non-preemptively.

Ni et al. [27] introduce process duplication to cope both with fail-stop and silent errors. Their pioneering paper contains many interesting results. It differs from this work in that they limit themselves to perfectly parallel applications while we investigate per task speedup profiles that obey Amdahl's law. More recently, Subasi et al. [34] proposed a software-based selective replication of task-parallel applications used for both fail-stop and silent errors. In contrast, this work (i) considers dependent tasks such as found in applications consisting of linear workflows; and (ii) proposes an optimal dynamic programming algorithm to solve the selective replication *and* checkpointing problem. Combining replication with checkpointing has also been proposed in [30], [42], [17] for HPC platforms, and in [23], [38] for grid computing.

VII. CONCLUSION

In this paper, we have studied the combination of checkpointing and replication to minimize the execution time of linear workflows in a failure-prone environment. We have introduced a sophisticated dynamic programming algorithm that solves the problem optimally, by determining which tasks to checkpoint and which tasks to replicate in order to minimize the total execution time. This dynamic programming algorithm

has been validated through extensive simulations that reveal the conditions in which checkpointing, replication, or both lead to improved performance. We have observed that the gain over the checkpoint-only approach is quite significant, in particular when checkpoint is costly and error rate is high.

Future work will address workflows whose dependence graphs are more complex than linear chains of tasks. Although an optimal solution seems hard to reach, the design of efficient heuristics that decide where to locate checkpoints and when to use replication would prove highly beneficial for the efficient and reliable execution of HPC applications on current and future large-scale platforms. We also plan to run some experiments on real applications in the near future. Finally, extending the approach to cope with both fail-stop and silent errors would be interesting, since both error sources are massively and simultaneously present on large-scale platforms.

Acknowledgements: This work has been partially supported by the Swiss Platform for Advanced Scientific Computing (PASC) project SPH-EXA.

REFERENCES

[1] G. Amdahl. The validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proceedings*, volume 30, pages 483–485. AFIPS Press, 1967.

[2] A. Benoit, A. Cavelan, F. Cappello, P. Raghavan, Y. Robert, and H. Sun. Identifying the right replication level to detect and correct silent errors at scale. Research report RR-9047, INRIA, 2017.

[3] A. Benoit, A. Cavelan, F. Ciorba, V. L. Fèvre, and Y. Robert. Combining checkpointing and replication for reliable execution of linear workflows. Research report RR-9152, INRIA, 2018.

[4] A. Benoit, A. Cavelan, Y. Robert, and H. Sun. Optimal resilience patterns to cope with fail-stop and silent errors. In *IPDPS*. IEEE, 2016.

[5] G. Bosilca, A. Bouteiller, E. Brunet, F. Cappello, J. Dongarra, A. Guermouche, T. Herault, Y. Robert, F. Vivien, and D. Zaidouni. Unified model for assessing checkpointing protocols at extreme-scale. *Concurrency and Computation: Practice and Experience*, 2013.

[6] E. S. Buneci. *Qualitative Performance Analysis for Large-Scale Scientific Workflows*. PhD thesis, Duke University, 2008.

[7] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir. Toward Exascale Resilience: 2014 update. *Supercomputing frontiers and innovations*, 1(1), 2014.

[8] H. Casanova, Y. Robert, F. Vivien, and D. Zaidouni. On the impact of process replication on executions of large-scale parallel applications with coordinated checkpointing. *Future Gen. Comp. Syst.*, 51:7–19, 2015.

[9] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985.

[10] S. P. Crago, D. I. Kang, M. Kang, R. Kost, K. Singh, J. Suh, and J. P. Walters. Programming models and development software for a space-based many-core processor. In *4th Int. Conf. onon Space Mission Challenges for Information Technology*, pages 95–102. IEEE, 2011.

[11] V. Cuevas-Vicenttín, S. C. Dey, S. Köhler, S. Riddle, and B. Ludäscher. Scientific workflows and provenance: Introduction and research opportunities. *Datenbank-Spektrum*, 12(3):193–203, 2012.

[12] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Comp. Syst.*, 22(3):303–312, 2006.

[13] S. Di, Y. Robert, F. Vivien, and F. Cappello. Toward an optimal online checkpoint solution under a two-level HPC checkpoint model. *IEEE Trans. Parallel & Distributed Systems*, 2016.

[14] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann. Combining partial redundancy and checkpointing for HPC. In *ICDCS*. IEEE, 2012.

[15] E. Elnozahy and J. Plank. Checkpointing for peta-scale systems: a look into the future of practical rollback-recovery. *IEEE Trans. Dependable and Secure Computing*, 1(2):97–108, 2004.

[16] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Survey*, 34:375–408, 2002.

[17] C. Engelmann, H. H. Ong, and S. L. Scorr. The case for modular redundancy in large-scale high performance computing systems. In *PDCN*. IASTED, 2009.

[18] C. Engelmann and B. Swen. Redundant execution of HPC applications with MR-MPI. In *PDCN*. IASTED, 2011.

[19] K. Ferreira, J. Stearley, J. H. I. Laros, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold. Evaluating the viability of process replication reliability for exascale systems. In *SC'11*. ACM, 2011.

[20] C. George and S. S. Vadhiyar. ADFT: An adaptive framework for fault tolerance on large scale systems using application malleability. *Procedia Computer Science*, 9:166 – 175, 2012.

[21] T. Hérault and Y. Robert, editors. *Fault-Tolerance Techniques for High-Performance Computing*, Computer Communications and Networks. Springer Verlag, 2015.

[22] G. Kandaswamy, A. Mandal, and D. A. Reed. Fault tolerance and recovery of scientific workflows on computational grids. In *Proc. of CCGrid'2008*, pages 777–782, 2008.

[23] T. Leblanc, R. Anand, E. Gabriel, and J. Subhlok. VolpexMPI: An MPI Library for Execution of Parallel Applications on Volatile Nodes. In *16th European PVM/MPI Users' Group Meeting*, pages 124–133. Springer-Verlag, 2009.

[24] R. Lucas, J. Ang, K. Bergman, S. Borkar, W. Carlson, L. Carrington, G. Chiu, R. Colwell, W. Dally, J. Dongarra, et al. Top ten exascale research challenges. *DOE ASCAC subcommittee report*, pages 1–86, 2014.

[25] R. E. Lyons and W. Vanderkulk. The use of triple-modular redundancy to improve computer reliability. *IBM J. Res. Dev.*, 6(2):200–209, 1962.

[26] D. P. Mehta, C. Shetters, and D. W. Bouldin. Meta-Algorithms for Scheduling a Chain of Coarse-Grained Tasks on an Array of Reconfigurable FPGAs. *VLSI Design*, 2013.

[27] X. Ni, E. Meneses, N. Jain, and L. V. Kalé. ACR: Automatic Checkpoint/Restart for Soft and Hard Error Protection. In *SC*. ACM, 2013.

[28] J. Plank, K. Li, and M. Puening. Diskless checkpointing. *IEEE Trans. Parallel Dist. Systems*, 9(10):972–986, 1998.

[29] M. W. Rashid and M. C. Huang. Supporting highly-decoupled thread-level redundancy for parallel programs. In *Proc. HPCA'2008*, pages 393–404. IEEE, 2008.

[30] B. Schroeder and G. A. Gibson. Understanding Failures in Petascale Computers. *Journal of Physics: Conference Series*, 78(1), 2007.

[31] L. Silva and J. Silva. Using two-level stable storage for efficient checkpointing. *IEE Proceedings - Software*, 145(6):198–202, 1998.

[32] J. Stearley, K. B. Ferreira, D. J. Robinson, J. Laros, K. T. Pedretti, D. Arnold, P. G. Bridges, and R. Riesen. Does partial replication pay off? In *FTXS*. IEEE, 2012.

[33] O. Subasi, J. Arias, O. Unsal, J. Labarta, and A. Cristal. Programmer-directed partial redundancy for resilient HPC. In *Computing Frontiers*. ACM, 2015.

[34] O. Subasi, G. Yalcin, F. Zyulkyarov, O. Unsal, and J. Labarta. Designing and Modelling Selective Replication for Fault-Tolerant HPC Applications. In *Proc. CCGrid'2017*, pages 452–457, May 2017.

[35] D. Talia. Workflow Systems for Science: Concepts and Tools. *ISRN Software Engineering*, 2013.

[36] S. Toueg and Ö. Babaoglu. On the optimum checkpoint selection problem. *SIAM J. Comput.*, 13(3):630–649, 1984.

[37] N. H. Vaidya. A case for two-level distributed recovery schemes. *SIGMETRICS Perform. Eval. Rev.*, 23(1):64–73, 1995.

[38] S. Yi, D. Kondo, B. Kim, G. Park, and Y. Cho. Using replication and checkpointing for reliable task management in computational grids. In *SC*. ACM, 2010.

[39] J. W. Young. A first order approximation to the optimum checkpoint interval. *Comm. of the ACM*, 17(9):530–531, 1974.

[40] J. Yu, D. Jian, Z. Wu, and H. Liu. Thread-level redundancy fault tolerant CMP based on relaxed input replication. In *ICCIT*. IEEE, 2011.

[41] G. Zheng, L. Shi, and L. V. Kale. FTC-Charm++: an in-memory checkpoint-based fault tolerant runtime for Charm++ and MPI. In *IEEE Int. Conf. on Cluster Computing*, pages 93–103, 2004.

[42] Z. Zheng and Z. Lan. Reliability-aware scalability models for high performance computing. In *Cluster Computing*. IEEE, 2009.

[43] Z. Zheng, L. Yu, and Z. Lan. Reliability-aware speedup models for parallel applications with coordinated checkpointing/restart. *IEEE Trans. Computers*, 64(5):1402–1415, 2015.